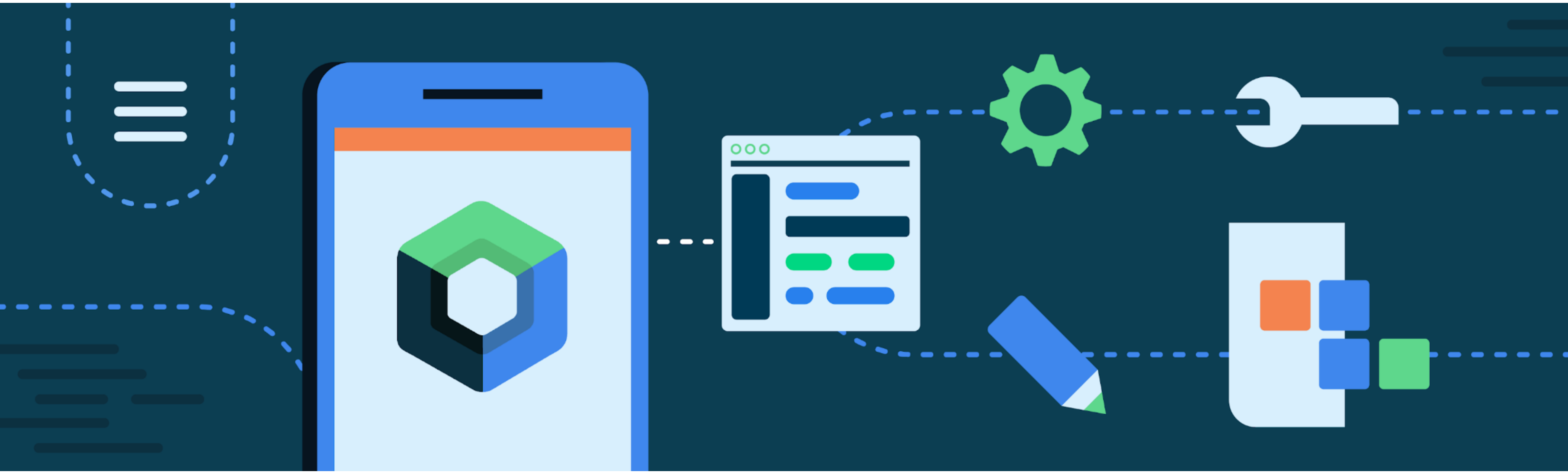
CMPS 312



UI Components and Layouts

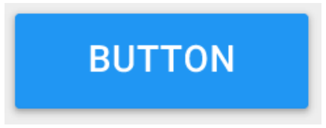
Dr. Abdelkarim Erradi

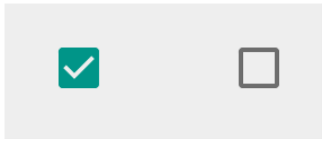
CSE@QU

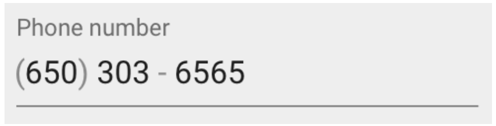
Outline

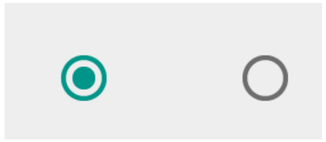
Examples are available @
cmps312-content\examples\05.ui-components-layouts

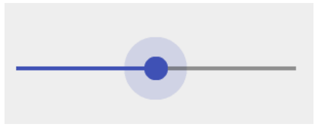
UI Components

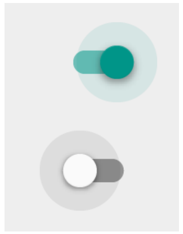
Button 

CheckBox 

TextField 

RadioButton 

Slider 

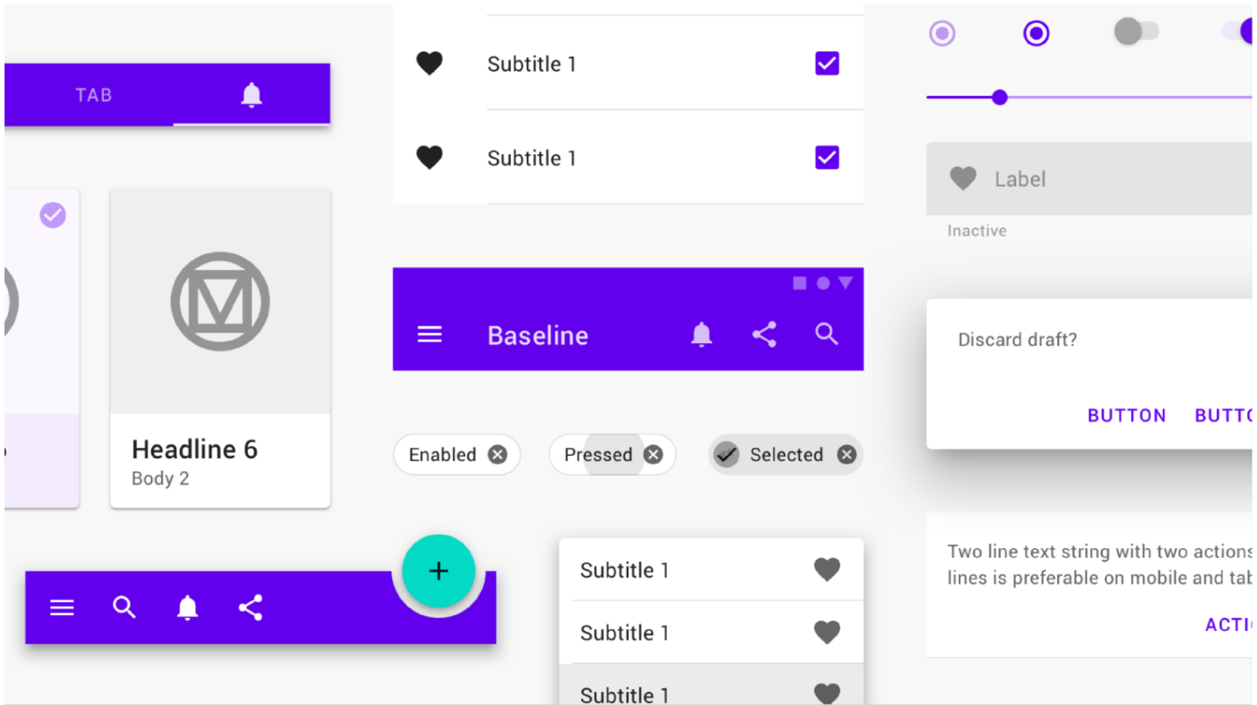
Switch 

Back

Full list available at this link

UI Components

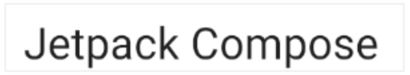
- Built-in Jetpack Compose UI Components follow Google Material design system https://material.io/
  - Provides a **consistent experience** across all platforms and applications (Android, Web, Flutter, iOS)



Text

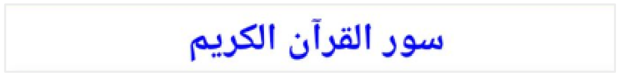
- **Text()** displays a text

```
Text(
    text = "Jetpack Compose",
    style = MaterialTheme.typography.h5
)
```



Jetpack Compose

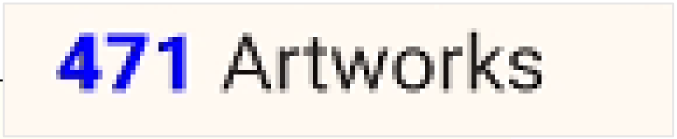
```
Text(
    text = "سور القرآن الكريم",
    textAlign = TextAlign.Center,
    modifier = Modifier.fillMaxWidth(),
    style = TextStyle(
        fontWeight = FontWeight.Bold,
        fontSize = 24.sp,
        color = Color.Blue,
        textDirection = TextDirection.Rtl
    )
)
```



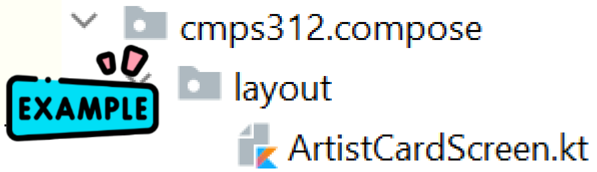
سور القرآن الكريم

Styled Text

- To set different styles within the same Text composable, use an **AnnotatedString**, a string annotated with styles using ***buildAnnotatedString()***

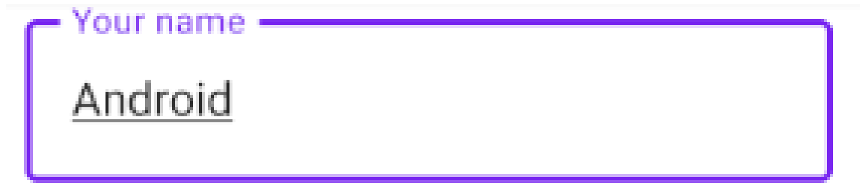

471 Artworks

```
Text(
    text = buildAnnotatedString {
        withStyle(
            style = SpanStyle(fontWeight = FontWeight.ExtraBold,
                              color = Color.Blue)
        ) {
            append("${artist.artWorksCount}")
        }
        withStyle(style = SpanStyle(fontWeight = FontWeight.Normal)) {
            append(" Artworks")
        }
    }
)
```



TextField

- **`TextField()`** collects input from the user. For more styling options, use **`OutlinedTextField()`**



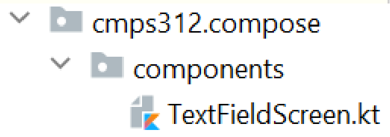
```
@Composable
fun NameEditor(name: String, onNameChange: (String) -> Unit) {
    OutlinedTextField(
        value = name,
        onValueChange = onNameChange,
        label = { Text("Your name") }
    )
}
```

Keyboard options

- TextField lets you set **keyboard configurations** options, such as the keyboard layout, or enable the autocorrect, capitalization, autoCorrect, keyboardType
- Also, it lets you to set a visual formatting of the input value, like replacing characters with * for passwords

```kotlin
@Composable
fun PasswordTextField() {
    var password by remember { mutableStateOf("") }

    OutlinedTextField(
        value = password,
        onValueChange = { password = it },
        label = { Text("Password") },
        visualTransformation = PasswordVisualTransformation(),
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password)
    )
}
```

EXAMPLE



```
cmps312.compose
  components
    TextFieldScreen.kt
```

Image

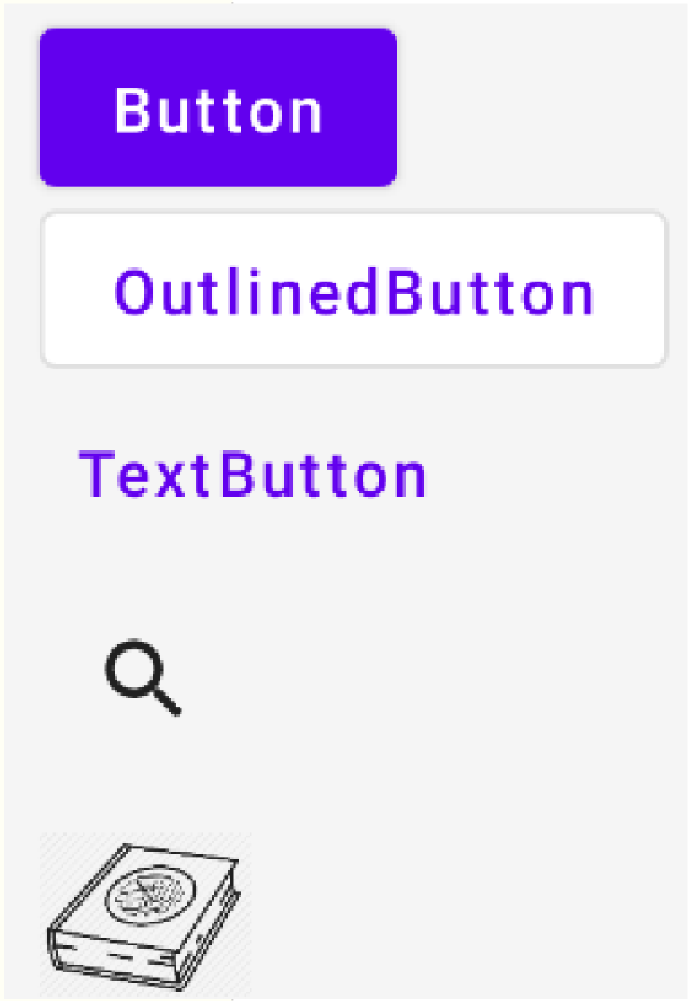
- Displays an image from the res/drawable folder

```
Image(painter =
    painterResource(R.drawable.img_compose_logo),
    contentDescription = "Jetpack compose logo",
    modifier = Modifier.height(300.dp))
```

Button

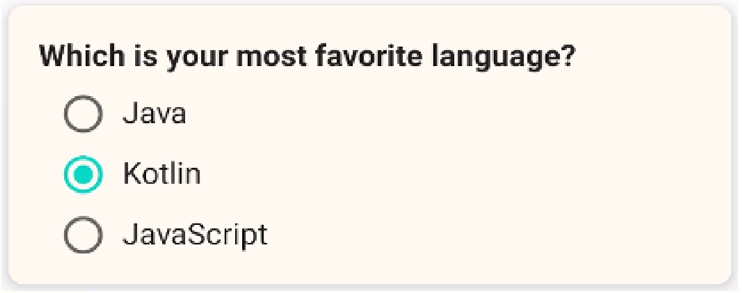
cmps312.compose
components
ButtonScreen.kt

```
Button(onClick = {}) {
    Text("Button")
}

OutlinedButton(onClick = {}) {
    Text("OutlinedButton")
}

TextButton(onClick = {}) {
    Text("TextButton")
}

// Search icons @ https://fonts.google.com/icons
IconButton(onClick = {}) {
    Icon(
        Icons.Outlined.Search,
        contentDescription = "Search",
    )
}

IconButton(onClick = {}) {
    Icon(painterResource(id = R.drawable.ic_quran), "Quran")
}
```



Radio Button

- A Radio Button is used to select a **single** option from a list of options


Which is your most favorite language?

- ○ Java
- ● Kotlin
- ○ JavaScript

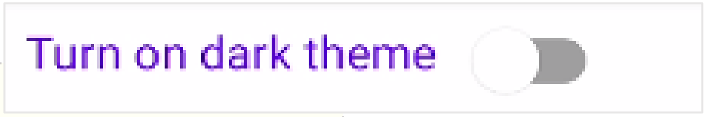
```
radioOptions.forEach { option ->
    Row(
        Modifier
            .fillMaxWidth()
            .selectable(
                selected = (option == selectedOption),
                onClick = { onOptionSelected(option) }
            )
            .padding(horizontal = 16.dp, vertical = 4.dp)
    ) {
        RadioButton(
            selected = (option == selectedOption),
            onClick = { onOptionSelected(option) }
        )

        Text(
            text = option,
            modifier = Modifier.padding(start = 8.dp)
        )
    }
}
```

EXAMPLE

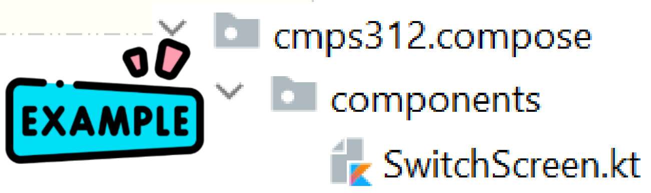
```
cmps312.compose
  components
    RadioButtonScreen.kt
```

Switch

- A Switch toggle the state of a single item on or off


Turn on dark theme

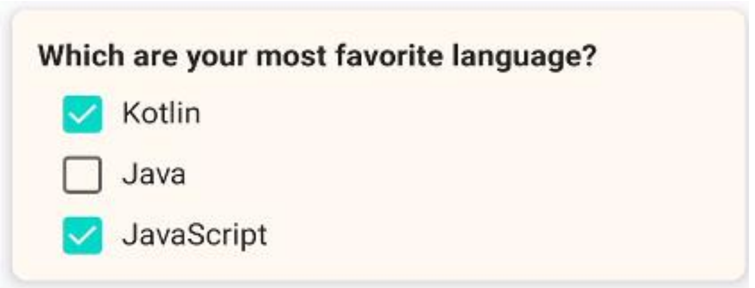
```
Row {
    Text(
        text = "Turn on dark theme",
        modifier = Modifier.padding(end = 8.dp)
    )
    Switch(
        checked = isDarkMode,
        onCheckedChange = { isDarkMode = it }
    )
}
```



cmps312.compose
components
SwitchScreen.kt

Checkbox

- Checkbox is used to represent two states i.e., either *checked* or *unchecked*
- When grouped multiple values can be selected


Which are your most favorite language?

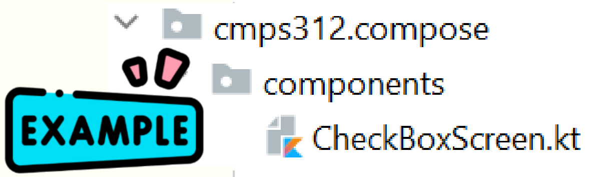
✓ Kotlin

☐ Java

✓ JavaScript

Kotlin (true)
Java (false)
JavaScript (true)

```
options.forEach { (option, isChecked) ->
    Row(
        Modifier
            .fillMaxWidth()
            .selectable(
                selected = isChecked,
                onClick = { onCheckedChange(option, !isChecked) }
            )
    ) {
        Checkbox(
            checked = isChecked,
            onCheckedChange = { onCheckedChange(option, it) }
        )
        Text(
            text = option,
            modifier = Modifier.padding(start = 8.dp)
        )
    }
}
```

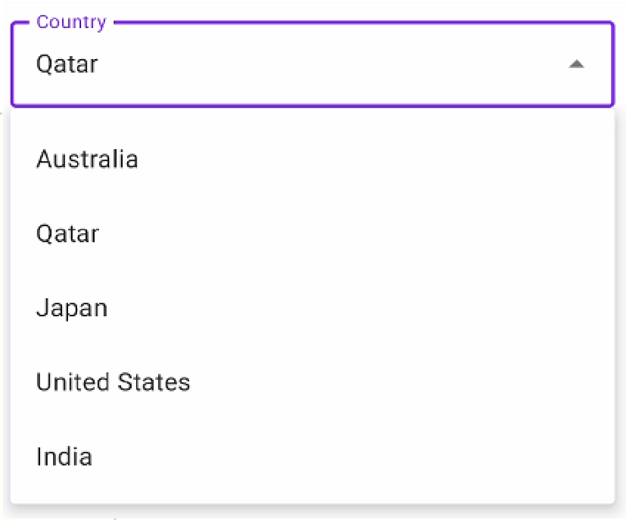


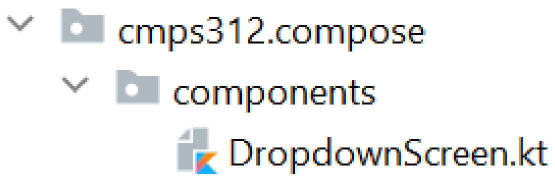
cmps312.compose
  components
    CheckBoxScreen.kt

Dropdown

- A Dropdown is used to select a single option from a list of options

```
DropdownMenu(
    expanded = expanded,
    onDismissRequest = { expanded = false },
) {
    options.forEach { option ->
        DropdownMenuItem(onClick = {
            expanded = false
            onSelectionChange(option)
        }) {
            Text(text = option)
        }
    }
}
```

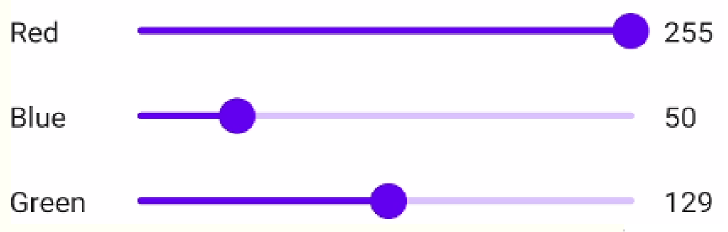


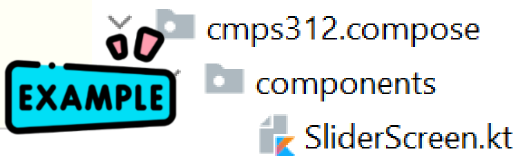


Slider

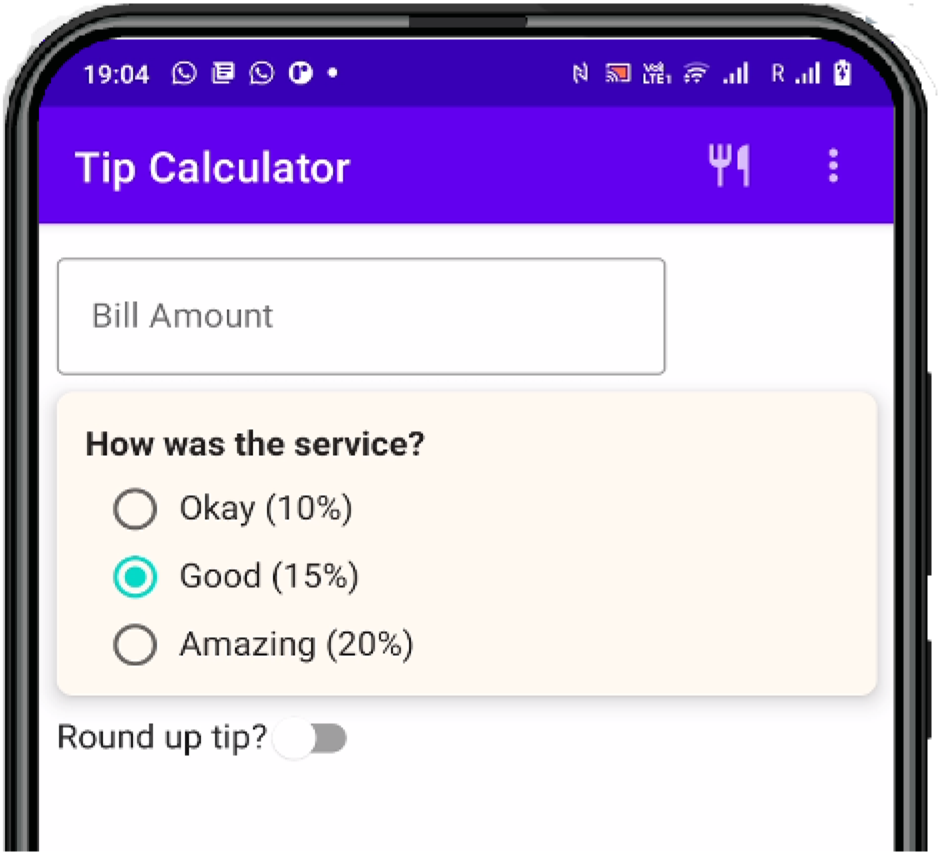
- Slider allows the user to select a value from a range along a bar. It is deal for adjusting settings such as volume and brightness.

```kotlin
fun ColorSlider(colorName: String, colorValue: Int, onValueChange: (Int) -> Unit) {
    Row {
        Text(text = colorName, modifier = Modifier.weight(1.5F))
        Slider(
            value = colorValue/100f,
            onValueChange = { onValueChange((it * 100).roundToInt()) },
            valueRange = 0.0f..2.55f,
            modifier = Modifier.weight(8F)
        )
        Text(
            text = "$colorValue",
            modifier = Modifier.weight(1F)
        )
    }
}
```





Tip Calculator Example



cmps312.compose
TipCalculator.kt

Layouts

https://developer.android.com/jetpack/compose/layouts/basics

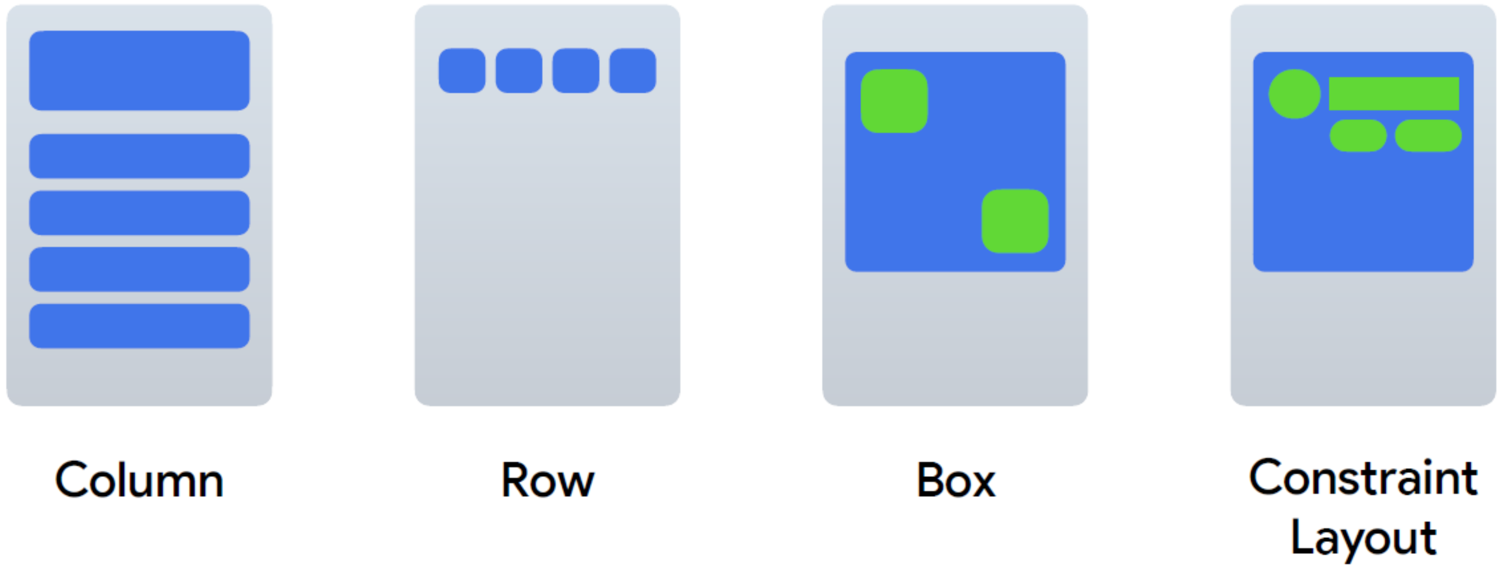
< Back

Responsive UI

- Layout automatically **controls** the **size** and **placement (position** and **alignment)** of UI elements to create a **Responsive UI**
  - Layouts provide an efficient way to **distribute space** among items while accommodating different screen sizes
  - Frees programmer from handling/hardcoding the sizing and positioning of UI elements
  - **Responsive UI =** When the screen is resized, the views reorganize themselves based on the rules of the layout

Layouts

- Use a Layout to **position** UI elements on the screen
  - Control of space distribution and alignment of items in a container
- **`Row`** - position elements horizontally
- **`Column`** - position elements vertically
- **`Box`** - position elements in the corners of the screen or stack them on top of each other
- Use [Constraint Layout](#)(self-study) for complex layouts



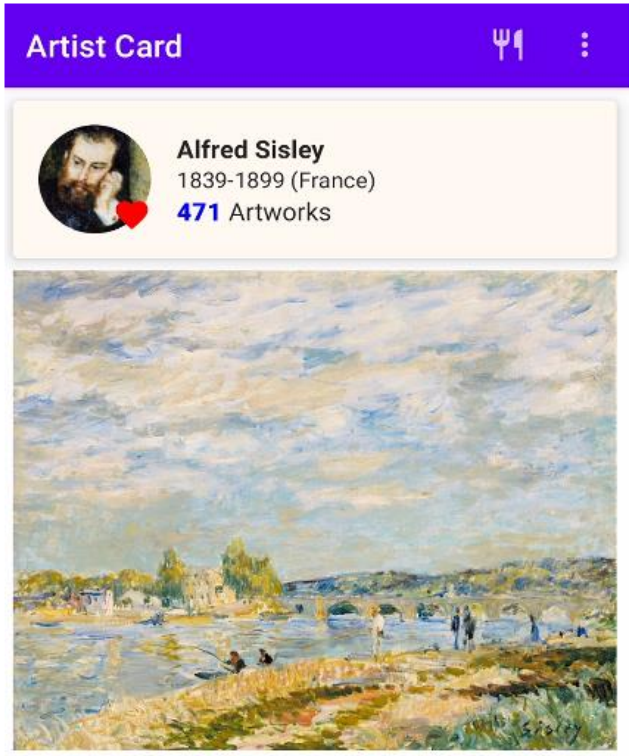
Composing Multiple Layouts

- Compose multiple basic layouts to create a more complex screen
  - Use vertical or horizontal **alignment** to change the position of elements
  - Use **modifiers** to style UI elements

EXAMPLE

```
cmps312.compose
  layout
    ArtistCardScreen.kt
```

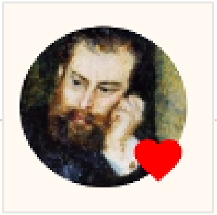
```
@Composable
fun ArtistCard(artist: Artist) {
    Column {
        Card {
            Row {
                Box {
                    Image(...)
                    Icon (...)
                }
                Column {
                    Text("${artist.lastName}")
                    Text("${artist.country}")
                }
            }
        }
        Image (...)     }
    }
}
```



Alfred Sisley
1839-1899 (France)
471 Artworks

Box Layout

- The children of the Box layout are **stacked** over each other
  - You can use the `align` modifier to specify the position of child element
  - E.g., the *Favorite* 💜 icon is overlayed over the image and aligned to the bottom right (`Alignment.BottomEnd`)



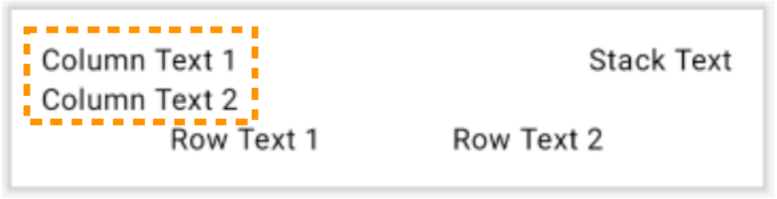
```
Box {
    Image(
        painter = painterResource(id = R.drawable.img_alfred_sisley),
        contentDescription = "${artist.firstName} ${artist.lastName}",
    )
    Icon(
        imageVector = Icons.Filled.Favorite,
        contentDescription = "Favorite",
        modifier = Modifier.align(Alignment.BottomEnd),
    )
}
```

Box Example (1 of 3)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```

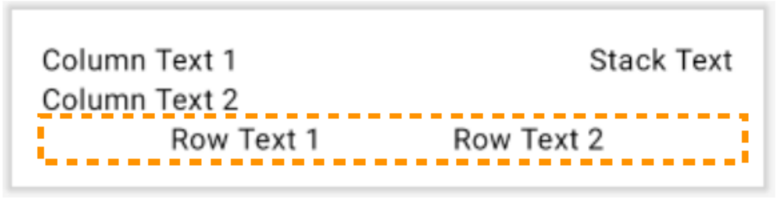
EXAMPLE

```
cmps312.compose
  layout
    BoxLayoutScreen.kt
```



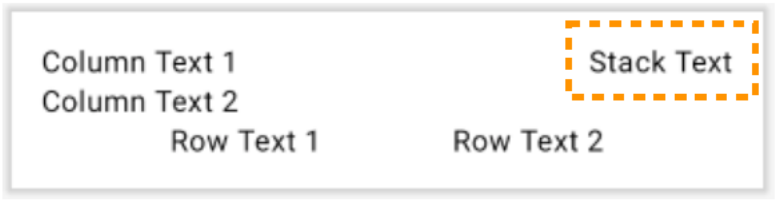
Box Example (2 of 3)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```



Box Example (3 of 3)

```
Box(modifier = Modifier.fillMaxWidth()) {
    Column(
        modifier = Modifier
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        Text("Column Text 1")
        Text("Column Text 2")

        Row(
            modifier = Modifier.fillMaxWidth(),
            horizontalArrangement = Arrangement.SpaceEvenly
        ) {
            Text(text = "Row Text 1")
            Text(text = "Row Text 2")
        }
    }
    Text(
        "Stack Text",
        modifier = Modifier
            .align(Alignment.TopEnd)
            .padding(end = 16.dp, top = 16.dp)
    )
}
```

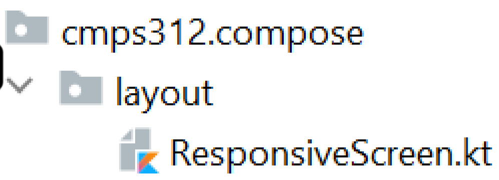


Surface & Card

- A **Surface** can hold only one child with an option to add a border and elevation
  - Add a layout inside Surface to position multiple elements
- A **Card** is a just a Surface with default parameters

Responsive Layout

EXAMPLE



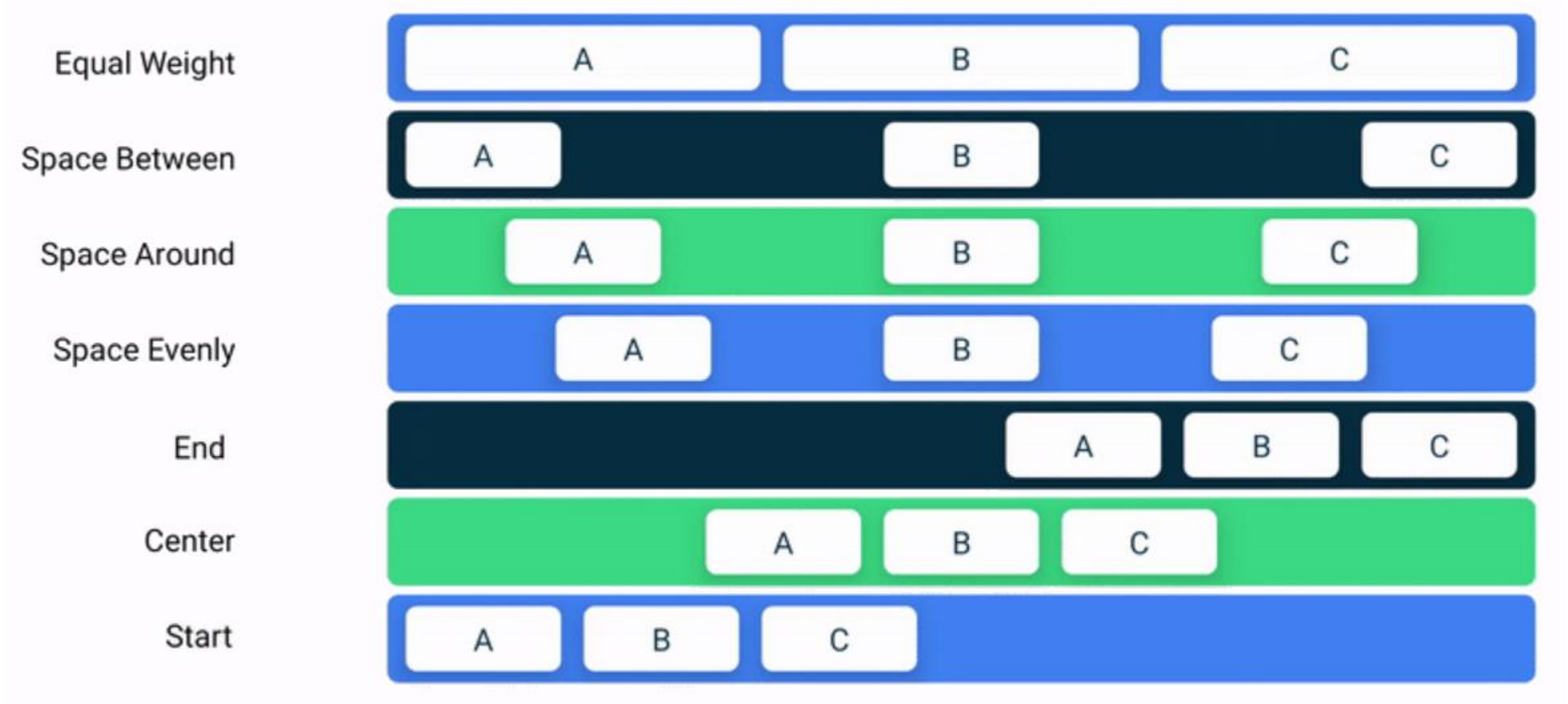
cmps312.compose
layout
ResponsiveScreen.kt

- Use the **`weight`** modifier in Row and Column layouts to change the **proportion** of the screen child elements will use
  - Distribute space among items in a container while accommodating different screen sizes
- `Modifier.`*`fillMaxWidth`*`()` fill available width
- `Modifier.`*`fillMaxHeight`*`()` fill available height
- `Modifier.`*`fillMaxSize`*`()` fill available width and height
- For more control use Constraint Layout (self-study) for complex scenarios

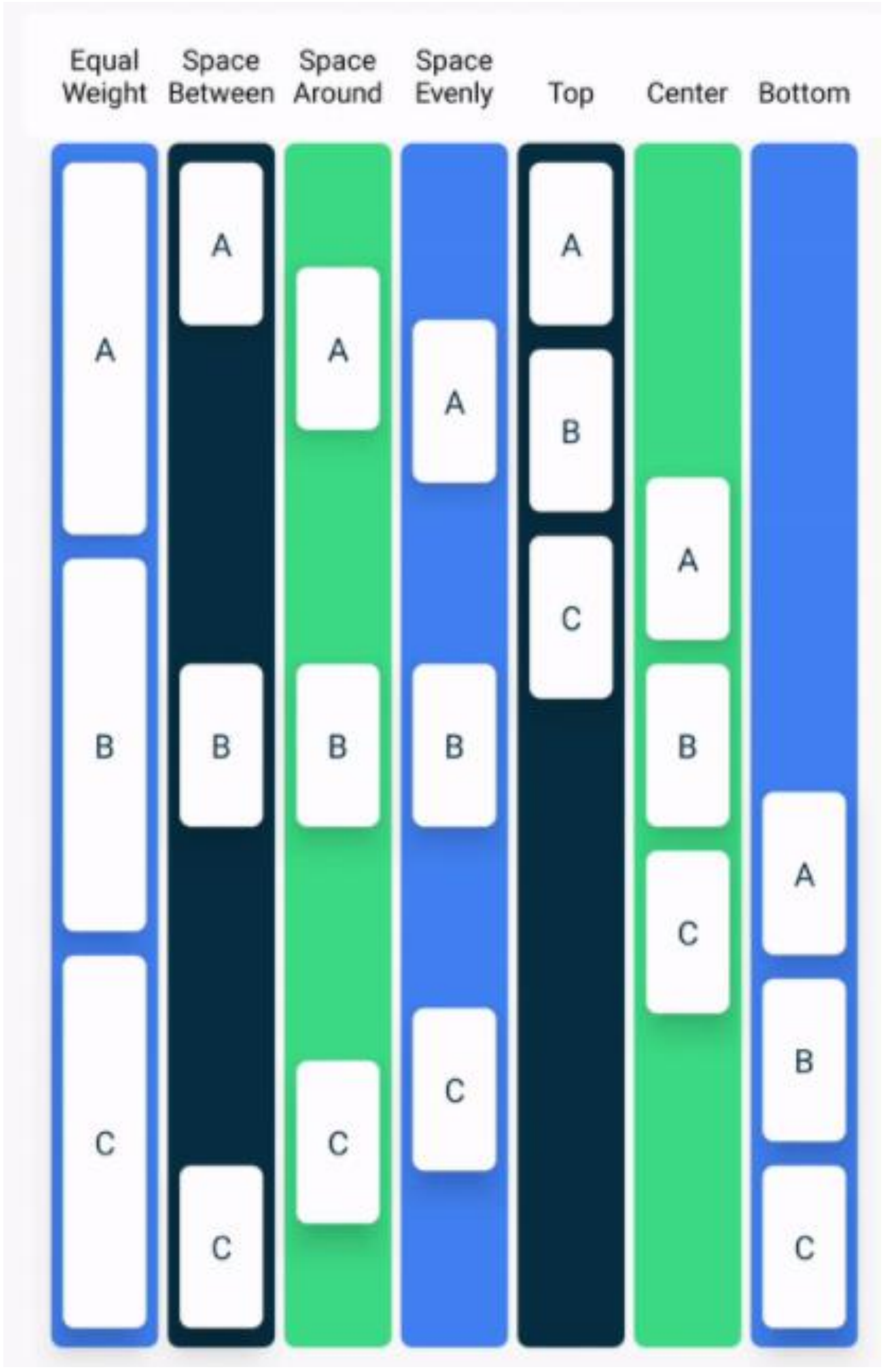
Responsive Layout using Arrangement & Alignment

- Layout parameters can be used to configure the alignment of the elements within it
- To set children's position within a Row, set the **`horizontalArrangement`** and **verticalAlignment** arguments
- For a Column, set the **`verticalArrangement`** and **`horizontalAlignment`** arguments

Horizontal Arrangement



- Distribute extra leftover free space along the horizontal axis
- You also you can use **spacedBy**(space: Dp) to specify the space between child elements
- More details available at thin [link]



Vertical Arrangement

- Distribute extra leftover free space along the vertical axis

Summary

- Use **TextField** - for collecting input from the user
- Use **RadioButtons** - for single item selection from a small set of options
- Use **Dropdown** - for single item selection from a larger set of options
- Use **Checkboxes** - for multi-item selection
- Use **Slider** - for picking a value from a range
- Use a Layout to position UI elements on the screen
  - **Row** - position elements horizontally
  - **Column** - position elements vertically
  - Use vertical or horizontal **alignment** to change the position of elements
  - Use **modifiers** to style UI elements

Resources

- Jetpack compose tutorial

https://developer.android.com/jetpack/compose/tutorial

- Jetpack compose Code Labs

https://developer.android.com/courses/pathways/compose

- Jetpack Compose Playground - UI component examples

https://foso.github.io/Jetpack-Compose-Playground/

https://github.com/Foso/Jetpack-Compose-Playground

https://github.com/Gurupreet/ComposeCookBook

- Compose Samples

https://github.com/android/compose-samples